CSN08x14

Scripting for Cybersecurity and Networks

Lecture 8: System programming

Lecture 8 - Topics

- System Programming
  - *OS, File System interaction*
  - *Calling/automating external processes*

Hash Password Cracking Script – Discussion

Dict_crack.py (Lab 3)

- **Dictionary Attack** on password hash
- Did you complete the script?

Discussion – efficiency

- How could we make script more efficient and more future-proof / easier to maintain?

```
# Script:  dict_crack.py
# Description: Cracks password hash using a dictionary attack.
# Author:  Petra L & Rich McF
# Modified: Sept 2018
import sys
import hashlib

# list of passwords
dic = ['123','1234','12345','123456','1234567','123456
        'password', 'qwerty','abc','abcd','abc123','11
        'monkey','arsenal','letmein','trustno1','drago
        'baseball','superman','iloveyou','starwars',
        'montypython','cheese','123123','football','batman']

# create list of corresponding md5 hashes using a list
hashes = [None for pwd in dic] ### replace None with y

# zip dic and hashes to create a dictionary (rainbow t
rainbow = {} ### replace empty dictionary with your fo


def dict_attack(passwd_hash):
    """Checks password hash against a dictionary of common passwords"""

    print (f'[*] Cracking hash: {passwd_hash}')


    passwd_found = None ### replace None with a look up u

    if passwd_found:
        print (f'[+] Password recovered: {passwd_found}')
    else:
        print (f'[-] Password not recovered')

def main():
    print('[dict_crack] Tests')
    passwd_hash = '4297f44b13955235245b2497399d7a93'
    dict_attack(passwd_hash)

if __name__ == '__main__':
        main()
```

List of common passwords

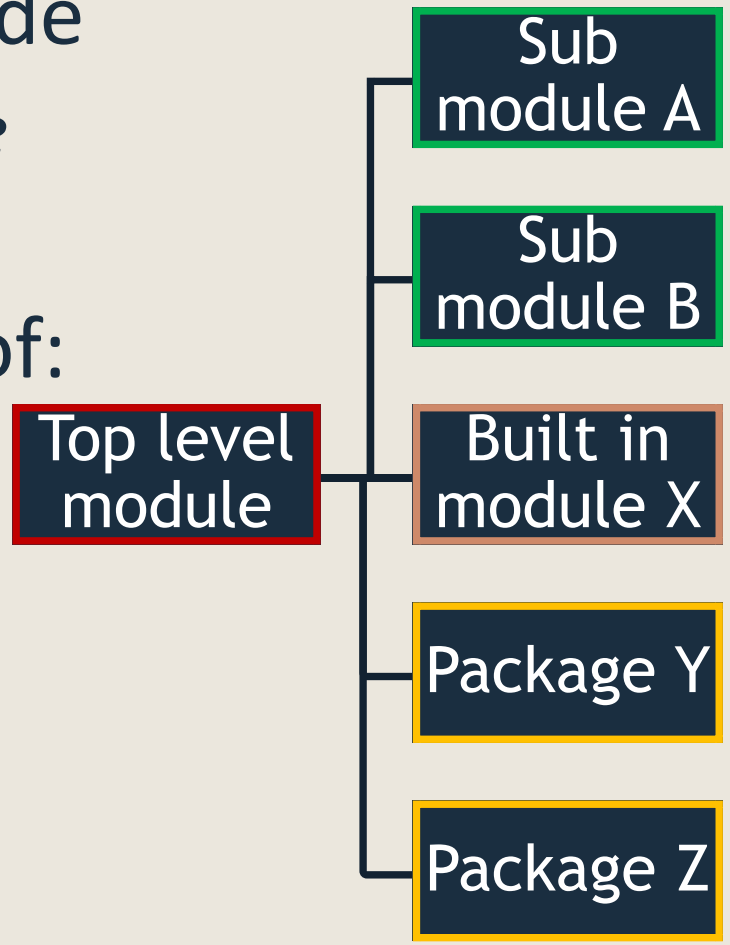
Create dictionary of hashes and passwords

Hash Signature Recovery function called with argument of password hash

Python Modules

Python Modules

- Each **Module** should contain **related** code
  - *such as functions and variables that serve a common purpose*
- Typically a Python application consists of:
  - *One top level module,*
  - *which imports*
    - (the programmer's own) sub modules
    - and publically available and/or built-in modules/packages.



Python Modules

- A **Module** is a Python **.py file** containing reusable code.

3 Types:

Script Modules	• Autonomous code which runs standalone. • Typically from command line, often with parameters and using arguments.
Function Library Modules	• Variable and function definitions only • Imported and then used by other code
Mixed use Modules	• Can be run as a standalone script • Or imported as a library of functions

Mixed Use Python Modules

1 Run as a standalone Script:

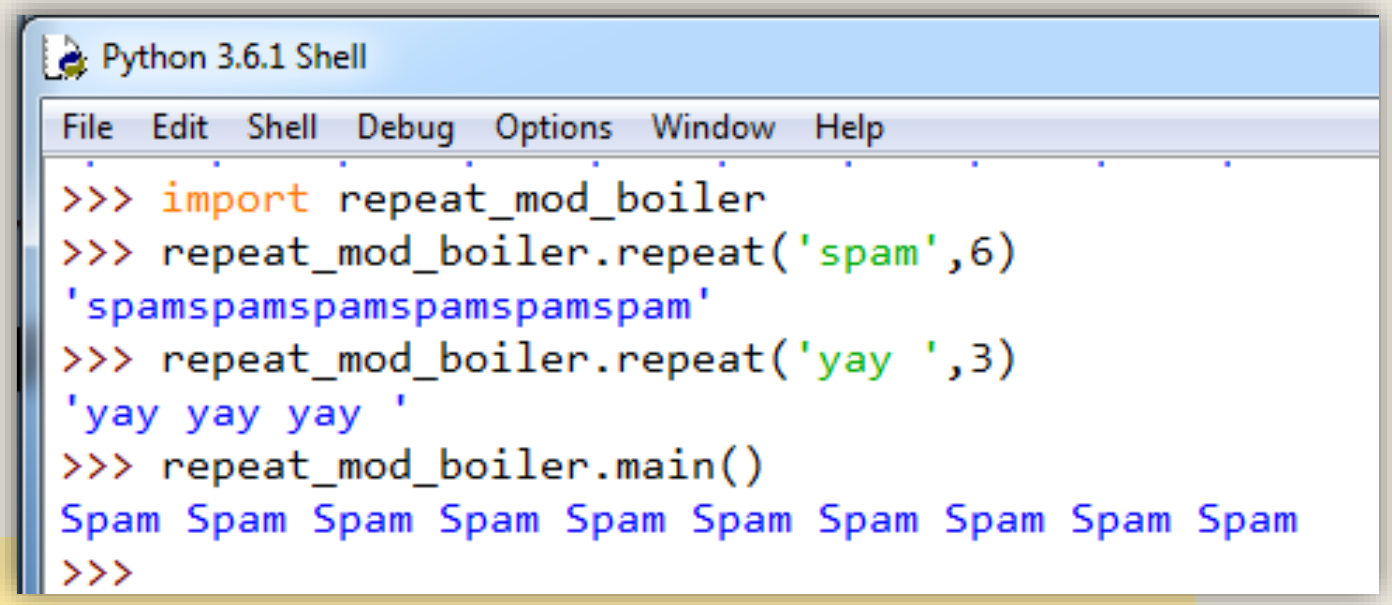
(Script could also be run from Win command line)

(in IDLE) Run > Run Module (F5)

```
========== RESTART: F:/Dropbox/CSN08114 Python/repeat_mod_boiler.py ==========
Spam Spam Spam Spam Spam Spam Spam Spam Spam Spam
>>>
```

2 Use as a library of functions:

import module or specific functions then use functions



How does it work?

repeat_mod_boiler.py - F:/Dropbox/CSN08114 Python/repeat_mod_boiler.py (3.6.1)

File Edit Format Run Options Window Help

```
def repeat(s, times):
    """ concatenates the input string s with itself times times
    and returns the concatenated string"""

    result = s * times
    return result

def main():
    # call the function with some test values
    print(repeat("Spam ",10))

# standard boilerplate to call the main() functi
if __name__ == '__main__':
    main()
```

Special main() Function
Can be used to test library of functions or
to call Module code functions as
a script

Boilerplate Code
This is the code that runs!!
Calls main() if run as a script, but does not run any code if imported as library.

Operating System Interaction with **sys**

sys: Python System Interaction

Python Interpreter System Module

`sys` **utility module –** variables and methods which interact with the Python Interpreter

- **sys.argv** Command line arguments passed to a Python script
- **sys.exit** terminate Python
- **sys.modules** Modules currently loaded
- **sys.path** Python module search path (loaded from PYTHONPATH)

```
>>> import sys
>>> dir(sys)
['__displayhook__', '__doc__', '__excepthook__', '__interactivehook__', '__loader__', '__name__', '__package__', '__spec__', '__stderr__', '__stdi
n__', '__stdout__', '_clear_type_cache', '_current_frames', '_debugmallocstats', '_enablelegacywindowsfsencoding', '_getframe', '_git', '_home', '
_xoptions', 'api_version', 'argv', 'base_exec_prefix', 'base_prefix', 'builtin_module_names', 'byteorder', 'call_tracing', 'callstats', 'copyright
', 'displayhook', 'dllhandle', 'dont_write_bytecode', 'exc_info', 'excepthook', 'exec_prefix', 'executable', 'exit', 'flags', 'float_info', 'float
_repr_style', 'get_asyncgen_hooks', 'get_coroutine_wrapper', 'getallocatedblocks', 'getcheckinterval', 'getdefaultencoding', 'getfilesystemencodee
rrors', 'getfilesystemencoding', 'getprofile', 'getrecursionlimit', 'getrefcount', 'getsizeof', 'getswitchinterval', 'gettrace', 'getwindowsversio
n', 'hash_info', 'hexversion', 'implementation', 'int_info', 'intern', 'is_finalizing', 'last_traceback', 'last_type', 'last_value', 'maxsize', 'm
axunicode', 'meta_path', 'modules', 'path', 'path_hooks', 'path_importer_cache', 'platform', 'prefix', 'set_asyncgen_hooks', 'set_coroutine_wrappe
r', 'setcheckinterval', 'setprofile', 'setrecursionlimit', 'setswitchinterval', 'settrace', 'stderr', 'stdin', 'stdout', 'thread_info', 'version',
 'version_info', 'warnoptions', 'winver']
```

Command line arguments

- When running a script from the command line, we may want to give it additional arguments
- These are stored as **sys.argv** (as long as sys is imported)
- and are therefore accessible to the script itself

Command line arguments: simple example

```
# argecho.py
# very simple command line argument demo for lecture
# http://www.diveintopython.net/scripts_and_streams/command_line_arguments.html

import sys

for arg in sys.argv:
    print (arg)
```

sys.argv[0] is the script itself
sys.argv[1] is the first argument
sys.argv[2] ..the 2nd argument
etc

```
Administrator: Windows Command Processor
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserv

C:\Windows\System32>f:

F:\>cd "F:\Dropbox\CSN08114 Python"

F:\Dropbox\CSN08114 Python>python argecho.py
argecho.py

F:\Dropbox\CSN08114 Python>python argecho.py and more
argecho.py
and
more

F:\Dropbox\CSN08114 Python>python argecho.py "and more"
argecho.py
and more
```

Using sys.argv

- How could we use sys.argv for dict_crack?

```
def main():
    print('[dict_crack] Tests')
    if len(sys.argv)==1:
        sys.argv.append('4297f44b13955235245b2497399d7a93')
    dict_attack(sys.argv[1])
```

If we pass a hash as parameter, it cracks that hash
If no parameter is passed when script run, uses the hash defined by sys.argv.append

Using sys.argv

- How can we require at least one argument to be passed at runtime?

```
def main():
    # checks if any arguments have been passed to the script and
    # tells you how to pass them
    if len(sys.argv) <2:
        print ('Usage: args.py argument1 [argument2] [argument3] ...')
        sys.exit(1)

    print ('[print_args] Tests')
    print_args(sys.argv[1:])
```

sys.exit(1) exits Python by raising SystemExit exception; performs cleanup
1 indicates abnormal termination (0 “successful termination” is the default)

Operating System Interaction with **os**

Operating System Interaction

- We may want to interact with the operating system from within Python
- Python is platform independent

but OS differ:

	MS-DOS	Linux
Copies files	copy	cp
Moves files	move	mv
Lists files	dir	ls
Closes shell prompt	exit	exit
Deletes files	del	rm
Finds a string of text in a file	find	grep
Creates a directory	mkdir	mkdir
Displays current location	chdir	pwd
Changes directories	cd	cd
Displays the time	time	date
Displays or sets date	date	date
Shows amount of RAM in use	mem	free

https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/4/html/Step_by_Step_Guide/ap-doslinux.html

So how can we solve this issue?

os module: Operating System Interaction

- The os module is a portable interface to underlying os
- Platform Independent

File Handling Constants

```
>>> import os
>>> dir(os)
['DirEntry', 'F_OK', 'MutableMapping', 'O_APPEND', 'O_BINARY', 'O_CREAT', 'O_EXCL', 'O_NOINHERIT', 'O_RANDOM', 'O_RDONLY'
, 'O_RDWR', 'O_SEQUENTIAL', 'O_SHORT_LIVED', 'O_TEMPORARY', 'O_TEXT', 'O_TRUNC', 'O_WRONLY', 'P_DETACH', 'P_NOWAIT', 'P_N
OWAITO', 'P_OVERLAY', 'P_WAIT', 'PathLike', 'R_OK', 'SEEK_CUR', 'SEEK_END', 'SEEK_SET', 'TMP_MAX', 'W_OK', 'X_OK', '_Envi
ron', '__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__'
, '_execvpe', '_exists', '_exit', '_fspath', '_get_exports_list', '_putenv', '_unsetenv', '_wrap_close', 'abc', 'abort',
'access', 'altsep', 'chdir', 'chmod', 'close', 'closerange', 'cpu_count', 'curdir', 'defpath', 'device_encoding', 'devnul
l', 'dup', 'dup2', 'environ', 'errno', 'error', 'execl', 'execle', 'execlp', 'execlpe', 'execv', 'execve', 'execvp', 'exe
cvpe', 'extsep', 'fdopen', 'fsdecode', 'fsencode', 'fspath', 'fstat', 'fsync', 'ftruncate', 'get_exec_path', 'get_handle_
inheritable', 'get_inheritable', 'get_terminal_size', 'getcwd', 'getcwdb', 'getenv', 'getlogin', 'getpid', 'getppid', 'is
atty', 'kill', 'linesep', 'link', 'listdir', 'lseek', 'lstat', 'makedirs', 'mkdir', 'name', 'open', 'pardir', 'path', 'pa
thsep', 'pipe', 'popen', 'putenv', 'read', 'readlink', 'remove', 'removedirs', 'rename', 'renames', 'replace', 'rmdir', '
scandir', 'sep', 'set_handle_inheritable', 'set_inheritable', 'spawnl', 'spawnle', 'spawnv', 'spawnve', 'st', 'startfile'
, 'stat', 'stat_float_times', 'stat_result', 'statvfs_result', 'strerror', 'supports_bytes_environ', 'supports_dir_fd', '
supports_effective_ids', 'supports_fd', 'supports_follow_symlinks', 'symlink', 'sys', 'system', 'terminal_size', 'times',
'times_result', 'truncate', 'umask', 'uname_result', 'unlink', 'urandom', 'utime', 'waitpid', 'walk', 'write']
```

Current working directory

os.path module!

os module – quick intro

```
>>> import os
>>> os.name
'nt'
>>> os.getcwd()
'C:\\Program Files\\Python36'
>>> os.listdir('c:\\temp')
[ 'temp.txt', 'rubbish_slides.pdf' ]
```

OS independent - uses ls or dir

Platform Independent Constants

- use in code instead of hard coded for a platform
- E.g. os.curdir, os.sep, os.linesep, os.pathsep

```
>>> os.sep
'\\'
>>> os.linesep
'\r\n'
```

```
>>> os.curdir
'.'
```

Q: How could we print the files in the current dir (whatever that happens to be)?

os.path: file system interaction

- `os.path` utility sub-module – variables and methods which interact with the File System
- Portable interface to underlying File System

```
>>> dir(os.path)
['__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__'
, '__spec__', '_get_bothseps', '_getfinalpathname', '_getfullpathname', '_getvolumepathname', 'abspath',
'altsep', 'basename', 'commonpath', 'commonprefix', 'curdir', 'defpath', 'devnull', 'dirname', 'exists',
'expanduser', 'expandvars', 'extsep', 'genericpath', 'getatime', 'getctime', 'getmtime', 'getsize', 'isa
bs', 'isdir', 'isfile', 'islink', 'ismount', 'join', 'lexists', 'normcase', 'normpath', 'os', 'pardir',
'pathsep', 'realpath', 'relpath', 'samefile', 'sameopenfile', 'samestat', 'sep', 'split', 'splitdrive',
'splitext', 'splitunc', 'stat', 'supports_unicode_filenames', 'sys']
```

Split path into filename and extension

Absolute path

File System Path Interaction with os.path module

- **os.listdir(dir)** List the filenames in the directory path specified by argument passed in

```
>>> os.listdir(r'c:\temp')
[ 'temp.txt', 'rubbish_slides.pdf' ]
```

Q: What is the "r" before the string?

- **os.path.curdir** Returns current dir (relative path)

```
>>> os.path.curdir
'.'
```

Relative path

How can I find current path in absolute form?

- **os.path.abspath(path)** Given a path returns an absolute path

```
>>> os.path.abspath('.\\temp.txt')
'C:\\Python\\temp.txt'
```

Absolute path

File System Path Interaction with os.path module

- **os.path.join(dir, filename)** Put dir and filename together to make a path (uses os.sep)

```
>>> os.path.join(os.path.curdir, 'temp.txt')
'.\\temp.txt'
```

- **os.path.split(path)** Given a path, returns tuple containing path and filename parts

```
>>> os.path.split('c:\\tmp\\temp.txt')
('c:\\tmp', 'temp.txt')
```

- **os.path.dirname(path)** Given a path, returns directory part

```
>>> os.path.dirname(r'c:\tmp\temp.txt')
'c:\\tmp\'
```

- **os.path.basename(path)** Given a path, returns filename part

```
>>> os.path.basename(r'c:\temp\temp.txt')
'temp.txt'
```

File System Path Interaction with os.path module

- **os.path.splitext(file)** filename path and extension separately as tuple

```
>>> os.path.splitext('c:\\temp\\temp.txt')
('c:\\temp\\temp', '.txt')
```

Q:

os.path.basename('geo_ip.py')

os.path.dirname('geo_ip.py')

os.path.dirname(r'c:\Program files\python37\geo_ip.py')

os.path.basename(r'c:\Program files\python37\geo_ip.py')

os.path.split(r'c:\Program files\python37\geo_ip.py')

os.path.splitext(r'c:\Program files\python37\geo_ip.py')

File System Path Interaction with os.path module

- **os.path.exists(path)** Given a path returns True/False if path exists/not

```
>>> os.path.exists('c:\\temp')
True
>>> os.path.isfile('c:\\temp\\nonexistant.exe')
False
```

- **os.path.isdir(path) os.path.isfile(path)** Given a path returns True if path is a dir/file

```
>>> os.path.isdir('c:\\temp')
True
>>> os.path.isfile('c:\\temp')
False
```

External Processes: **subprocess** module

Calling External Processes

- **subprocess module**

```
>>> import subprocess
>>> dir(subprocess)
['CREATE_NEW_CONSOLE', 'CREATE_NEW_PROCESS_GROUP', 'CalledProcessError', 'CompletedProcess', 'DEVNULL'
, 'Handle', 'PIPE', 'Popen', 'STARTF_USESHOWWINDOW', 'STARTF_USESTDHANDLES', 'STARTUPINFO', 'STDOUT',
'STD_ERROR_HANDLE', 'STD_INPUT_HANDLE', 'STD_OUTPUT_HANDLE', 'SW_HIDE', 'SubprocessError', 'TimeoutExp
ired', '_PLATFORM_DEFAULT_CLOSE_FDS', '__all__', '__builtins__', '__cached__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__', '_active', '_args_from_interpreter_flags', '_clea
nup', '_mswindows', '_optim_args_from_interpreter_flags', '_time', '_winapi', 'builtins', 'call', 'che
ck_call', 'check_output', 'errno', 'getoutput', 'getstatusoutput', 'io', 'list2cmdline', 'msvcrt', 'os
', 'run', 'signal', 'sys', 'threading', 'time', 'warnings']
```

subprocess.Popen
replaces old popen()
functions in os module

Calling External Processes

Example help:

- **subprocess.call(cmd)**

```
>>> help(subprocess.call)
Help on function call in module subprocess:

call(*popenargs, timeout=None, **kwargs)
    Run command with arguments.  Wait for command to complete or
    timeout, then return the returncode attribute.

    The arguments are the same as for the Popen constructor.  Example:

    retcode = call(["ls", "-l"])
```

Calling External Processes: Example 1

```
>>> import subprocess
>>> cmd = 'ipconfig'
>>> output = subprocess.check_output(cmd)
>>> print (output.decode())
```

Q: Why do we need to decode the output??

```
Windows IP Configuration


Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . : napier.ac.uk
   Link-local IPv6 Address . . . . . : fe80::95ea:ed60:22b8:bde1%12
   IPv4 Address. . . . . . . . . . . : 146.176.164.159
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 146.176.164.254

rk Adapter VMnet1:

ffix  . :
   Link-local IPv6 Address . . . . . : fe80::a9d3:fb00:2b15:5673%15
   IPv4 Address. . . . . . . . . . . : 192.168.59.1
   Subnet Mask                       : 255.255.255.0
```

```
>>> print (output)
b'\r\nWindows IP Configuration\r\n\r\n\r\nEthernet adapter Local Area Connection:\r\n\r\n   Connection
-specific DNS Suffix  . : napier.ac.uk\r\n   Link-local IPv6 Address . . . . . : fe80::95ea:ed60:22b8:
bde1%12\r\n   IPv4 Address. . . . . . . . . . . : 146.176.164.159\r\n   Subnet Mask . . . . . .
```

Calling External Processes: Example 2

```
ipaddr='192.168.1.1'
proc = subprocess.Popen('ping ' + ipaddr,
                        stdout= subprocess.PIPE,
                        stderr= subprocess.PIPE,
                        shell=True)
# Get stdout, stderr from ping process
out, err = proc.communicate()
print (out.decode())
```

```
Pinging 192.168.1.1 with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.168.1.1:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

Practical Lab

- Webpage scraping script

Python Webpage Scraping Script

Webpage Scraping – find the webpage links with a regex

`webpage_getlinks.py`

lab07

```
# Script:   webpage_getlinks.py
# Desc:     Basic web site info gathering and analysis script.
#           From a URL gets page content, and parses out hyperlinks.
# Modified: Oct 2017

import sys, re
import webpage_get

def print_links(page):
    ''' find all hyperlinks on a webpage passed in as input and print '''
    print ('[*] print_links()')
    # regex to match on hyperlinks
    links = re.findall(r'\<a.*href\=.*http\:.+', page.decode())
    # sort and print the links
    links.sort()
    print (f'[+] {len(links)} HyperLinks Found:')
    for link in links:
        print(link)

def main():
    # temp testing url argument
    sys.argv.append(r'http://www.napier.ac.uk')

    # Check args
    if len(sys.argv) != 2:
        print ('[-] Usage: webpage_getlinks URL')
        return

    # Get the web page
    page = webpage_get.wget(sys.argv[1])
    # Get the links
    print_links(page)

if __name__ == '__main__':
        main()
```

Hyperlinks regex

Gets webpage content

Search for webpage hyperlinks

Python Webpage Scraping Script

Webpage Scraping – find the webpage links with a regex

`webpage_getlinks.py`

lab07

```
# Script:   webpage_getlinks.py
# Desc:     Basic web site info gathering and analysis script.
#           From a URL gets page content, and parses out hyperlinks.
# Modified: Oct 2017

import sys, re
import webpage_get

def print_links(page):
    ''' find all hyperlinks on a webpage passed in as input and print '''
    print ('[*] print_links()')
    # regex to match on hyperlinks
    links = re.findall(r'\<a.*href\=.*http\:.+', page.decode())
    # sort and print the links
    links.sort()
    print (f'[+] {len(links)} HyperLinks Found:')
    for link in links:
        print(link)

def main():
    page = webpage_get.wget(sys.argv[1])
    # Get the links
    print_links(page)

if __name__ == '__main__':
    main()
```

```
======= RESTART: F:/Dropbox/CSN08114 Python/webpage_getlinks_start.py =======
[*] print_links()
[+] 3 HyperLinks Found:
<a href="http://my.napier.ac.uk" target="_blank">
<a href="http://staff.napier.ac.uk" target="_blank">
<a href="http://www.napier.ac.uk/about-us/events/pg-information-evening-16-november"
id="ctl19_largediamondmiddlecontent_0_ctaLink">
>>>
```

Python Webpage Scraping Script

Webpage Scraping – find the webpage links with a regex

How can we improve the regex?

```
def print_links(page):
    ''' find all hyperlinks on a webpage passed in as input and print '''
    print ('[*] print_links()')
    # regex to match on hyperlinks
    links = re.findall(r'\<a.*href\=.*http\:.+', page.decode())
    # sort and print the links
    links.sort()
    print (f'[+] {len(links)} HyperLinks Found:')
    for link in links:
        print(link)
```

Hyperlink - how to only match on link itself?

```
======= RESTART: F:/Dropbox/CSN08114 Python/webpage_getlinks_start.py =======
[*] print_links()
[+] 3 HyperLinks Found:
http://my.napier.ac.uk
http://staff.napier.ac.uk
http://www.napier.ac.uk/about-us/events/pg-information-evening-16-november
```